M2 ISTR - Vérification et Validation

Model Checking

Julien Brunel, ONERA
Julien.Brunel@onera.fr

Plan

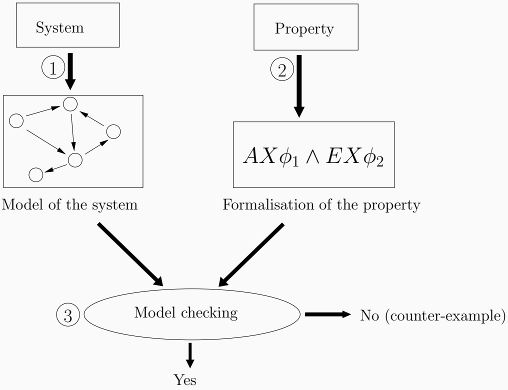
Introduction

Formal Methods

- Techniques based on mathematical methods to reason in a rigourous way
- Used in the design and validation of critical systems (railways, aeronautics, space, automotive)
- Costly (in terms of time and expertise) but errors and bugs are even more!
- Allow to have guarantees by proof

Model checking

1. Building of a formal model of the system
2. Formal expression of the properties to check (derived from the specification or from requirements)
3. Answer the question : *Does the model of the system satisfy the properties?*



Model checking

- Step 1 can be done by hand, or automatically.
  The system can be a simple program, an hardware architecture, or the abstraction of a more complex system, made of IT components and non-IT components (hydraulics for instance).

- Step 2 must be done by hand, and may need some expertise on the property language.

- Step 3 is in principle entirely automatic.

Advantages and drawbacks of model checking

Advantages

- can be used in early phases of development cycle
- automatic approach
- exhaustive exploration of the states of the system
- nice expressiveness (lots of properties can be expressed)
- efficiency according to the data structures

Limits

- needs formalisation
- expression of properties is non trivial
- finite number of states
- state explosion problem

Mitigate the state explosion problem

- efficient data structures : Binary Decision Diagram (BDD)
- abstract the model to decrease the number of states
- partial order reduction: do not consider several times executions that are equivalent for the satisfaction of the desired property
- induction : allows to represent in a finite way infinite structures
- ...

History of model checking

1977	Pnueli proposes to use temporal logic
1981	Model checking of CTL par Clarke et al., Sifakis et al.
1980-1990	Many theoretical results
1990-2000	Huge performance improvements
	Extensions : probabilities, real-time, infinite structures
2000-...	MC adopted by main chip marker corporations (*e.g.* Intel)
	Starting of software model checking (Microsoft)
	ACM Paris Kanellakis Award 1998 et 2005
2007	Turing Award to Clarke, Sifakis et Emerson
2010-...	new SAT-based algorithms

In practice

- Check properties of electronic circuits (Intel, Motorola, IBM, etc.)
- Check the absence of bugs, or find bugs in software (*software model checking*)
    - on Scade programs
    - on C code (BLAST from Berkeley, SLAM from Microsoft)
    - on Java code (JavaPathFinder)
    - on ByteCode, binary, ...
- Analyse the dependability of a system (AltaRica du LaBri/Dassault)
- Check the correctness of distributed systems (TLA+ used for instance by AWS)

Expression of the properties to check

Non temporal properties
Property about the value of variables or the data structure

- *The value of the integer variable* x *is greater than* y.
- *The array is sorted.*

⇒ out of the scope of model checking

Temporal Properties
Temporal aspects can have various forms

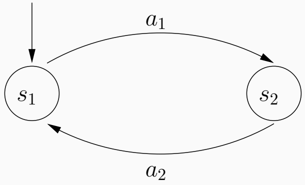
- *If a process requests to be executed, the OS will execute it eventually.*
- *It is always possible to go back to the initial state.*
- *Each time a failure is detected, an alarm is launched.*
- *Each time an alarm is launched, a failure has been detected earlier.*

Formal semantics of systems

Transition system

Definition (Transition system (TS))

- a set S of states
- a set $I \subseteq S$ of initial states
- a set L of labels
- a transition relation $\rightarrow \subseteq S \times L \times S$



Notation

$$s_1 \xrightarrow{a} s_2 \quad \stackrel{def}{=} \quad (s_1, a, s_2) \in \rightarrow$$
$$s_1 \rightarrow s_2 \quad \stackrel{def}{=} \quad \exists a \in L. s_1 \xrightarrow{a} s_2$$

Transition system (symbolic definition)

- States can be defined by variables
- Transitions can be defined by variable updates

A (very) simple resource allocator

```
VAR
request : boolean;
state : {ready,busy};
INIT
state = ready
TRANS
if (state = ready & request)
then state' = busy
else state' = ready || state' = busy
```

Terminology

We find different terms for very close concepts:

- Kripke models/structures in logic (model theory)
- State machine in software engineering
- Automata
  - in language theory,
  - or to model control structures at a higher level than TS (e.g., with variables)

Main differences between variants

- Finite of infinite number of states
- Determinism
- Label on states and/or transitions

Why so many similar frameworks?

- Historical reason

History of automata

- 1940s : to model neurons...
- 1960s : languages, computability
- 1970s : systems models
- 1980s : model checking

- Different scientific communities
- Finite automata: simple formalism, limited expressiveness, efficient algorithms
- Many results in various domains
- Many extensions : pushdown automata, automata with data structures (integers, ...), timed automata, Petri Nets

Properties to check on a transition system

Categories of properties

- Safety *Something bad never happens*
- Liveness *Something good will happen eventually*

- Accessibility *A given state can be reached*
- Invariance *If a given property is true before a transition, it is still true after this transition*
- Fairness *Transitions that are executable are executed eventually*

Formal property languages

Need for a property language

We want to express formally these kinds of properties.

What properties for this system?

```
VAR
  request : boolean;
  state : {ready,busy};
INIT
  state = ready
TRANS
  if (state = ready & request)
  then state' = busy
  else state' = ready || state' = busy
```

Propositional logic (syntax)

Definition (Syntax)
Given a set P of atomic propositions, the language of propositional logic is defined by :

- If $p \in P$ then p is a formula
- If A and B are formulas, then
  - $\neg A$ is a formula, $A \wedge B$ is a formula

Definition (Semantics)
A model, or valuation, for a formula A is a function $V : P \to \{true, false\}$ which associates each atomic proposition with a truth value (V is a line in the truth table).

$$
\begin{array}{lll}
V \models p & \text{iff} & V(p) \\
V \models \neg A & \text{iff} & V \not\models A \\
V \models A_1 \wedge A_2 & \text{iff} & V \models A_1 \text{ and } V \models A_2
\end{array}
$$

Remark
Define Boolean connectives $\vee$ and $\Rightarrow$ in terms of $\neg$ and $\wedge$.

Propositional logic (axiomatics)

Definition (Axiomatics)
Axioms

- $A_1 \Rightarrow (A_2 \Rightarrow A_1)$ Ax1
- $(A_1 \Rightarrow (A_2 \Rightarrow A_3)) \Rightarrow ((A_1 \Rightarrow A_2) \Rightarrow (A_1 \Rightarrow A_3))$ Ax2

Inference rule

- $\dfrac{A_1 \quad A_1 \Rightarrow A_2}{A_2}$ (Modus Ponens)

Valid formulas and theorems

Valid formula
A formula A is valid ($\models A$) if it is true for every valuation :

$$\models A \qquad \text{iff} \qquad \forall V \quad V \models A$$

Theorem
A formula A is a theorem ($\vdash A$) if it is an axiom or it is obtained by applying inference rules to axioms..

Exercise
Prove that $A \Rightarrow A$ is valid, and then prove that it is a theorem.

Valid formulas and theorems

Valid formula
A formula A is valid ($\models A$) if it is true for every valuation :

$$\models A \qquad \text{iff} \qquad \forall V \quad V \models A$$

Theorem
A formula A is a theorem ($\vdash A$) if it is an axiom or it is obtained by applying inference rules to axioms..

Exercise
Prove that $A \Rightarrow A$ is valid, and then prove that it is a theorem.

Definition (Correctness and completeness)

- A deduction system is correct if every theorem is valid.
- It is complete if every valid formula is a theorem.

Decision procedure

To know if a formula is valid (or satisfiable), there are different methods.

- the simplest : truth table
- many algorithms have been developed recently with the aim of efficiency
- method that will be useful for temporal logics : tableaux method
  Goal : build a model of a formula, if there is one. It is important to make sure the method is complete (if it does not produce a model, then there does not exist any).

Expressiveness of propositional logic

Try to express in propositional logic:

- *Function* `compute_position` *returns a correct result if functions* `gps` *and* `measure_speed` *return correct results.*
- *At least two of these three functions return a correct result.*
- *Each level 1 function returns a correct result if all the level 2 functions (on which it depends) return a correct result.*
- *After an incorrect result of function* `gps`*, function* `compute_position` *returns a result that stays incorrect for the whole system execution.*

First order logic

Definition
First order logic extends propositional logic with

- variables $x_1, x_2, \ldots$
- quantifiers $\exists, \forall$ on variables
- functions on variables (`succ` if we reason on integers)
- predicates which replace propositions, and which apply to terms (variables or function applications) ($\leqslant$ for instance):

$$\forall x. \forall y. \exists z. \quad \leqslant (x, z) \Rightarrow \quad \leqslant (succ(y), z)$$

First order logic is more expressive than propositional logic but it is undecidable.

Temporal logics

Temporal logics extend propositional logic to express dynamic behaviours instead of static properties.

- p will be true eventually.
- p will always be true.
- p is always followed by q.
- there exists an execution that will satisfy p.
- $\dots$

Linear Temporal Logic (LTL)

Definition (Syntax)
Given a set P of atomic propositions, the syntax of LTL is defined by :

- If $p \in P$ then p is a formula
- If A and B are formulas, then
  - $\neg A$ is a formula, $A \wedge B$ is a formula
  - $\mathrm{X}\ A$ is a formula, $A\ \mathrm{U}\ B$ is a formula

- $X\ A$: A will be true in the next state
- $A_1\ \mathrm{U}\ A_2$: A_1 will remain true until A_2 becomes true

To define in terms of the previous operators

- $\mathrm{F}\ A$: A will be true at some instant in the future
- $\mathrm{G}\ A$: A will always be true

Definition (Semantics)
A model is an infinite sequence $\sigma \in S^{\omega}$ of states $(s_0, s_1, \ldots)$ with a valuation function $V : S \to 2^P$.

$$
\begin{array}{lll}
\sigma, i \models p & \text{iff} & p \in V(\sigma_i) \\
\sigma, i \models \neg A & \text{iff} & \sigma, i \not\models A \\
\sigma, i \models A_1 \wedge A_2 & \text{iff} & \sigma, i \models A_1 \text{ and } \sigma, i \models A_2
\end{array}
$$

Linear Temporal Logic (LTL)

Definition (Semantics)
A model is an infinite sequence $\sigma \in S^{\omega}$ of states $(s_0, s_1, \ldots)$ with a valuation function $V : S \to 2^P$.

$$
\begin{array}{lll}
\sigma, i \models p & \text{iff} & p \in V(\sigma_i) \\
\sigma, i \models \neg A & \text{iff} & \sigma, i \not\models A \\
\sigma, i \models A_1 \wedge A_2 & \text{iff} & \sigma, i \models A_1 \text{ and } \sigma, i \models A_2 \\
\sigma, i \models A_1 \,\mathrm{U}\, A_2 & \text{iff} & \exists i' \geqslant i \quad \text{such that} \quad \sigma, i' \models A_2 \;\; \text{and} \\
 & & \forall\, i'' \in \mathbb{N} \quad \text{if} \quad i \leqslant i'' < i' \quad \text{then} \quad \sigma, i'' \models A_1
\end{array}
$$

Definition (Semantics)
A model is an infinite sequence $\sigma \in S^\omega$ of states $(s_0, s_1, \ldots)$ with a valuation function $V : S \to 2^P$.

$$
\begin{array}{lll}
\sigma, i \models p & \text{iff} & p \in V(\sigma_i) \\
\sigma, i \models \neg A & \text{iff} & \sigma, i \not\models A \\
\sigma, i \models A_1 \wedge A_2 & \text{iff} & \sigma, i \models A_1 \text{ and } \sigma, i \models A_2 \\
\sigma, i \models A_1 \,\mathrm{U}\, A_2 & \text{iff} & \exists i' \geqslant i \quad \text{such that} \quad \sigma, i' \models A_2 \text{ and} \\
 & & \forall\, i'' \in \mathbb{N} \quad \text{if} \quad i \leqslant i'' < i' \quad \text{then} \quad \sigma, i'' \models A_1 \\
\sigma, i \models \mathrm{X}\, A & \text{iff} & \ldots
\end{array}
$$

Expressiveness of LTL

Try to express in LTL

- *p will be true at least once.*
- *Each time p is true, q will be true later on*
- *p is true at most once*
- *p is true exactly twice*
- *p will only be true after q*
- *When p is true, there is an execution on which q will be true, and an execution in which r will be true*

Computation-Tree Logic (CTL)

Definition (Syntax)
Given a set P of atomic propositions, CTL syntax is defined as follows:

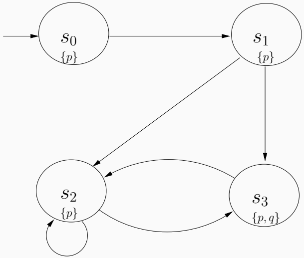
- If $p \in P$ then p is a formula
- If A and B are formulas, then
  - $\neg A$ is a formula, $A \wedge B$ is a formula
  - $\mathbf{E}\mathrm{X}\ A$ is a formula, $\mathbf{E}[A\ \mathrm{U}\ B]$ is a formula, $\mathbf{A}[A\ \mathrm{U}\ B]$ is a formula
- $\mathbf{E}\mathrm{X}\ A$: there exists a successor state satisfying A
- $\mathbf{E}[A_1\ \mathrm{U}\ A_2]$ / $\mathbf{A}[A_1\ \mathrm{U}\ A_2]$: there exists / all paths starting from the current state (that) satisfy(ies) $A_1\ \mathrm{U}\ A_2$

To define in terms of the previous operators :

- $\mathbf{A}\mathrm{X}\, A$: all the successors of the current state satisfy A
- $\mathbf{A}\mathrm{G}\, A$: A will always be true (in all the paths that start from the current state)
- $\mathbf{E}\mathrm{G}\, A, \quad \mathbf{A}\mathrm{F}\, A, \quad \mathbf{E}\mathrm{F}\, A$

Definition (CTL model)
A CTL model is a Kripke structure $(S, I, \rightarrow, V)$, où

- S is a set of states
- $I \subseteq S$ the set of initial states
- $\rightarrow \subseteq S \times S$ is the transition relation
- $V : S \rightarrow 2^P$ is a function mapping each state to the set of atomic propositions that are true in this state



Definition (Semantics)

$$
\begin{array}{lll}
s \models p & \text{iff} & p \in V(s) \quad \text{where} \quad p \in P \\
s \models \neg A & \text{iff} & s \not\models A \\
s \models A_1 \wedge A_2 & \text{iff} & s \models A_1 \quad \text{and} \quad s \models A_2 \\
 & & \\
s \models \mathbf{E}\mathrm{X}\, A & \text{iff} & \exists s' \in S \text{ such that } s \to s' \quad \text{and} \quad s' \models A \\
 & & \\
s \models \mathbf{A}[A_1 \;\mathrm{U}\; A_2] & \text{iff} & \forall \sigma \in \mathit{Paths}(s) \quad \exists i \in \mathbb{N} \quad \text{such that} \quad \sigma_i \models A_2 \\
 & & \text{and} \quad \forall j \in \mathbb{N} \quad \text{if} \quad 0 \leqslant j < i \quad \text{then} \quad \sigma_j \models A_1 \\
s \models \mathbf{E}[A_1 \;\mathrm{U}\; A_2] & \text{iff} & \exists \sigma \in \mathit{Paths}(s) \quad \exists i \in \mathbb{N} \quad \text{such that} \quad \sigma_i \models A_2 \\
 & & \text{and} \quad \forall j \in \mathbb{N} \quad \text{if} \quad 0 \leqslant j < i \quad \text{then} \quad \sigma_j \models A_1
\end{array}
$$

CTL semantics

Given $M = (S, I, \rightarrow, V)$ a model and A a CTL formula,

$$M \models A \qquad \text{iff} \qquad \forall s \in I \quad s \models A$$

LTL and CTL standard connectives

$$\mathrm{F}\,A \stackrel{def}{=} (\neg A)\,\mathrm{U}\,A$$
$$\mathrm{G}\,A \stackrel{def}{=} \neg\mathrm{F}\,\neg A$$

$$\mathbf{A}\mathrm{X}\,A \stackrel{def}{=} \neg\mathbf{E}\mathrm{X}\,\neg A$$
$$\mathbf{E}\mathrm{F}\,A \stackrel{def}{=} \mathbf{E}[\neg A\,\mathrm{U}\,A]$$
$$\mathbf{A}\mathrm{F}\,A \stackrel{def}{=} \mathbf{A}[\neg A\,\mathrm{U}\,A]$$
$$\mathbf{E}\mathrm{G}\,A \stackrel{def}{=} \neg\mathbf{A}\mathrm{F}\,\neg A$$
$$\mathbf{A}\mathrm{G}\,A \stackrel{def}{=} \neg\mathbf{E}\mathrm{F}\,\neg A$$

Comeback to LTL

Satisfaction of an LTL formula by a model
Given $M = (S, I, \to, V)$ a model and A an LTL formula,

$$M \models A \qquad \text{iff} \qquad \forall \sigma \in \mathsf{Paths}(M), \quad \sigma, 0 \models A$$

Theoretical results about LTL et CTL

Theorem
LTL and CTL are decidable. They both have correct and complete axiomatic systems.

Expressive power of two logics

Let L_1 and L_2 two logics having the same semantic models.

$L_1 \leqslant L_2$ (L_2 is more expressive than $L_1 ss$) if for any $A_1 \in L_1$, there is $A_2 \in L_2$ s.t. the models satisfying A_1 are the same as the models satisfying A_2.

Expressive power of two logics

Let L_1 and L_2 two logics having the same semantic models.

$L_1 \leqslant L_2$ (L_2 is more expressive than $L_1 ss$) if for any $A_1 \in L_1$, there is $A_2 \in L_2$ s.t. the models satisfying A_1 are the same as the models satisfying A_2.

Expressive power of LTL and CTL

Do we have LTL $\leqslant$ CTL or CTL $\leqslant$ LTL ?